Servicios Web: Estado del Arte y Frameworks de Desarrollo

José Abraham Rodríguez López (j.abraham8@hotmail.com)
Facultad de Informática – Universidad Politécnica de Madrid

Abstract *— Los Servicios Web son una de las tecnologías más utilizadas y más conocidas actualmente. Una gran cantidad de empresas están realizando desarrollos o utilizan los Servicios Web activamente. Esta tecnología dispone de unas características y funcionalidades muy interesantes. En este documento se hace un análisis de su estado del arte: su arquitectura, su pila de protocolos, las especificaciones adicionales… También se realiza un estudio y valoración personal de los dos frameworks más utilizados actualmente en el mercado para el desarrollo de Servicios Web con la plataforma Java.*

Palabras Clave — Apache Axis 2, Java, JAX-WS, Servicios Web.

MOTIVACIONES

El objetivo del presente documento es hacer una introducción al estado del arte de esta tecnología y analizar el proceso de generación de Servicios Web con los dos frameworks más utilizados del mercado para los desarrollos con Java.

OBJETIVOS ESPECÍFICOS

- Estudiar la arquitectura de un Servicio Web.
- Estudiar la pila de protocolos y analizar cuáles son los más importantes en los que se basa.
- Conocer las especificaciones adicionales existentes y las posibilidades que nos ofrecen.
- Conocer las principales ventajas que nos ofrecen los Servicios Web
- Saber en qué situaciones pueden resultarnos de utilidad.
- Analizar las ventajas e inconvenientes de Apache Axis 2 como herramienta de desarrollo de Servicios Web. Así como el proceso de generación de dicho servicio y del cliente.
- Analizar las ventajas e inconvenientes de JAX-WS como herramienta de desarrollo de Servicios Web. Así como el proceso de generación de dicho servicio y del cliente.
- Realizar una comparativa de las dos herramientas.
- Conocer las posibilidades a la hora de afrontar el desarrollo de un Servicio Web asíncrono.

ALCANCE

La finalidad del presente documento es introducir al lector en la tecnología subyacente bajo los Servicios Web, por tanto no se entrará en detalle a analizar ni los protocolos utilizados por esta tecnología, ni su arquitectura, ni las especificaciones adicionales, quedando estas como objetivo para futuras investigaciones.

De la misma forma se hace un análisis de las posibilidades que las dos herramientas estudiadas nos proporcionan para la generación de los Servicios Web y del código del cliente. El estudio se centrará en dicho proceso y su complejidad, saliéndose del ámbito de este documento el estudio de otras funcionalidades provistas, así como el estudio de su eficiencia y otras características.

Finalmente se hace un pequeño análisis de las posibilidades a la hora de afrontar el desarrollo de un Servicio Web asíncrono, pero tampoco se entrará a analizar cuál de estas posibilidades es mejor, más sencilla de desarrollar o más eficiente en lo que a diferentes aspectos se refiere.

INTRODUCCIÓN

El W3C define un "servicio web" como "un sistema software diseñado para soportar interacción máquina-a-máquina sobre una red. Tiene una interfaz descrita en formato procesable por una máquina (WSDL). Otros sistemas interactúan con el servicio web utilizando mensajes SOAP de la manera indicada en su especificación, típicamente transportados utilizando HTTP con serialización de XML en conjunto con otros estándares relacionados con la Web" [16]. De esta definición se deduce que, como su nombre indica, la finalidad de esta tecnología es permitir que una funcionalidad sea accesible a través de la red por otras máquinas como un servicio. Para ello se ayuda de una serie de estándares que permiten realizar la publicación, la búsqueda, la invocación y la composición de dicha funcionalidad. Adicionalmente, existen una serie de especificaciones que permiten configurar el servicio web para proveer al cliente de distintas calidades de servicio.

Los servicios web también se pueden definir como "aplicaciones autocontenidas, modulares, distribuidas y

dinámicas que pueden ser descritas, publicadas, localizadas o invocadas sobre la red para crear productos, procesos o cadenas de suministro. Estas aplicaciones pueden ser locales, distribuidas o basadas en la Web. Los servicios web son construidos en la cima de estándares abiertos como TCP/IP, HTTP, Java, HTML y XML" [20]. Si bien esta definición no aporta mucho sobre la anterior, si se destaca el carácter distribuido de los servicios web, de forma que se toma conciencia de la importancia de factores como la independencia de lenguaje o de plataforma si queremos obtener una implementación distribuida realmente eficaz.

Los servicios web son resultado de la continua evolución y mejora de la web. Han evolucionado rápido y se espera que cambien los paradigmas de uso y desarrollo de software, promoviendo la reusabilidad de software a través de la red y proveyendo funcionalidad sofisticada, rápida y flexible a través de la composición de servicios [21].

La gran ventaja de esta tecnología es su independencia, ya que soporta cualquier lenguaje de programación, además de cualquier entorno. Así, puede ser adoptada por cualquier tipo de organización, sea cual sea el lenguaje que utilice para sus desarrollos. Esta característica combinada con el hecho de que permite publicar funcionalidades ya desarrolladas abre una nueva puerta hacía el futuro, al permitir la colaboración, composición y comunicación de aplicaciones que por ser implementadas en lenguajes diferentes, antes no era posible.

Otro aspecto clave es que aplicaciones que llevan ya muchos años desarrolladas podrán ser ahora accesibles a través de la web. Esto representa un nuevo mercado, de forma que las organizaciones podrán proveer la funcionalidad a sus clientes a través de servicios web, así como a sus colaboradores.

Los estándares más importantes en los que se soporta esta tecnología son los siguientes:

- Simple Object Access Protocol (SOAP): utilizado para la comunicación entre los servicios web.
- Web Service Definition Language (WSDL): que permite realizar la descripción del servicio.
- Universal Description, Discovery and Integration (UDDI): directorio en el cual se almacenan las descripciones de los servicios web.

ARQUITECTURA

Descripción

En su nivel básico, puede ser considerada una arquitectura cliente-servidor universal que permite comunicarse a diferentes sistemas sin usar librerías propietarias [22]. Esto provoca que se pueda ver una evolución de los sistemas orientados a objetos hacía los sistemas orientados a servicios, lo cual permite una mayor desacoplamiento, así como el enlazado dinámico de servicios [19], lo cual representa una evolución lógica de la orientación a objetos.

La arquitectura está formada básicamente por un conjunto de especificaciones, cada una de las cuales está pensada para resolver una necesidad diferente. De esta forma, se utiliza WSDL para realizar la especificación y descripción del servicio web, que posteriormente será publicado utilizando UDDI. El cliente realiza una búsqueda en base a la descripción del servicio, y una vez encontrado el que satisfaga sus necesidades, obtiene su especificación. Para la comunicación que se llevará a cabo entre el cliente y el proveedor del servicio ambos utilizarán SOAP sobre HTTP. Esta arquitectura se puede observar en la Figura1.

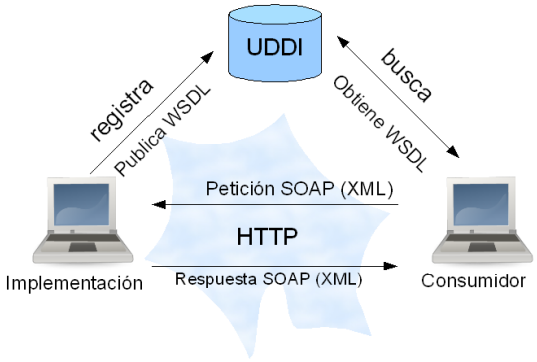

Figura 1: Arquitectura de Servicio Web

Dicha arquitectura está a su vez basada en SOA (Service Oriented Architecture) la cual permite crear aplicaciones distribuidas de forma que su funcionalidad pueda ser publicada, descubierta y enlazada para componer nuevos servicios que representen un valor añadido. La invocación de servicios se realiza a través de un endpoint y no a nivel de implementación, así las aplicaciones se vuelven más flexibles [23].

La arquitectura de los Servicios Web se soporta principalmente en tres estándares:

- **SOAP**: estándar basado en XML que permite definir los mensajes que serán intercambiados en un sistema de forma independiente al protocolo utilizado para su transporte. SOAP ha sido estandarizado por el W3C y está ampliamente aceptado por la industria.
- **WSDL**: estándar utilizado para la especificación de la interfaz del servicio, su localización y la forma de acceso. Al igual que SOAP ha sido estandarizado por el W3C.
- **UDDI**: utilizado para describir un registro basado en XML en el cual se almacena información sobre

Servicios Web. Incluye una serie de APIs para la búsqueda y publicación de servicios en los repositorios.

Pila de Protocolos

La pila de protocolos que componen la arquitectura de los servicios web está dividida en distintas capas. Dicha pila se puede observar en la Figura 2.

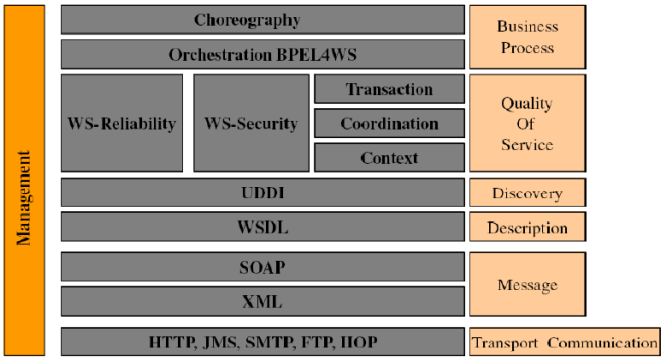

Figura 2: Pila de Protocolos de Servicios Web

A continuación se describirán las diferentes capas y estándares que componen que componen la pila de protocolos:

- **Transporte**: es la encargada de entregar los mensajes a las aplicaciones. Las especificaciones de servicios web permiten utilizar casi cualquier protocolo, aunque el más utilizado es HTTP.
- **Mensaje**: se especifica el formato que tendrán los mensajes intercambiados, su origen, destino, sistemas que los pueden procesar… Se utiliza XML para la codificación de los mensajes y SOAP para la comunicación de dichos mensajes utilizando un modelo de petición/respuesta.
- **Descripción**: el propósito de esta capa es definir una interfaz pública del servicio web, en la cual se expresan metadatos sobre las capacidades tanto funcionales como no funcionales del servicio. El lenguaje utilizado para tal fin es WSDL.
- **Descubrimiento**: esta capa permite la publicación de servicios en un repositorio común, así los usuarios pueden servicios que satisfagan sus necesidades. El estándar usado en esta capa es UDDI.
- **Calidad de servicio**: se hace cargo de los aspectos no funcionales tales como entrega fiable de mensajes, seguridad, gestión de transacciones…
- **Procesos de Negocio**: esta capa permite la composición de servicios ya existentes, así como la coordinación de sus interacciones, de forma que se puedan crear procesos de negocio complejos.

Especificaciones adicionales

Las especificaciones adicionales se encuentran enmarcadas dentro de la capa de Calidad de Servicio. Una lista completa de las especificaciones adicionales puede ser encontrada en [24]. A continuación se hace una breve descripción de las principales funcionalidades provistas por las más importantes especificaciones [25]:

- **Direccionamiento**: permite la identificación de los nodos que realizan el intercambio de mensajes a través de sus EPR (endpoint reference), independientemente del protocolo de transporte que utilicen.
- **Políticas**: define un framework para describir y combinar diferentes tipos de políticas dependiendo del acceso al servicio y las características de ejecución. Sirve para añadir características como reglas, comportamientos, requisitos, preferencias… La información provista por WS-Policy complementa las descripciones funcionales de WSDL.
- **Intercambio de Metadatos**: se definen protocolos para el intercambio de metadatos, que pueden ser políticas, descripciones WSDL… Esto ayuda a mejorar el proceso de descubrimiento de la descripción del servicio.
- **Seguridad**: propone una manera de utilizar técnicas ya existentes de seguridad de una forma interoperable. Algunos ejemplos son encriptación de XML, firma de XML… Básicamente se centra en los aspectos de identificación, autenticación, autorización, integridad, confidencialidad y no repudio.
- **Fiabilidad**: independientemente de que los protocolos de bajo nivel sean fiables o no, deben usarse otros mecanismos para garantizar la entrega de los mensajes o avisar de la causa del fallo.
- **Transacciones**: se garantiza que el resultado final de una serie de interacciones es coherente frente a errores de concurrencia, situaciones inesperadas. . . Para ello se utilizan protocolos de coordinación de las transacciones. La finalidad última es preservar las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad).
- **Composición**: se utilizar para especificar descripciones de composición de servicios. Dichas descripciones pueden ser ejecutadas y gestionadas de forma automática. El lenguaje utilizado para tales fines es WS-BPEL (Business Process Execution Language)
- **Notificación y Eventos**: permite la utilización de programación basada en eventos al proveer de un protocolo de suscripción a servicios y gestión de las suscripciones.

¿CUÁNDO UTILIZAR SERVICIOS WEB?

Los Servicios Web permiten la integración entre cualquier tipo de dispositivos, soportando cualquier lenguaje de programación sobre cualquier plataforma y cualquier red de transporte. Tecnologías como SOAP y XML son sencillas

de usar y flexibles. Todo esto provoca que los Servicios Web permitan una interacción dinámica sin precedentes [2]. A continuación se listan una serie de situaciones/motivos en los que utilizar Servicios Web:

- **Integración de sistemas heterogéneos**: por ejemplo cuando se tienen diferentes Sistemas Operativos o diferentes lenguajes de programación.
- **Entorno de cliente desconocido**: situación en la cual no se tiene conocimiento o/ni control sobre las aplicaciones cliente utilizadas para acceder al servicio.
- **Diferentes formatos cliente**: tales como diferentes navegadores, hojas de estilo, dispositivos inalámbricos y otras aplicaciones de negocio. Al devolverse el resultado de la invocación en XML este puede ser transformado a cualquiera de los formatos soportados por el cliente.
- **Otras aplicaciones de Servicios Web**: los Servicios Web pueden ayudar a alcanzar varios tipos de objetivos de negocio. Se pueden usar para resolver problemas tácticos, para gestionar aplicaciones heredadas reduciendo así los costes de desarrollo … También permiten optimizar tus procesos de negocio y mejorar las relaciones con el cliente.
- **Integración punto a punto**: comunicación de dos aplicaciones diferentes dentro de una misma organización.
- **Visión consolidada**: se provee una única visión de la información aunque esta provenga de varias fuentes.
- **Gestionar los activos legados**: el tiempo necesario para realizar modificaciones a software antiguo se ve reducido en gran tamaño.
- **Eliminar aplicaciones redundantes**: un servicio puede soportar varios tipos de clientes. Si se necesita llevar a cabo una acción que involucra varias aplicaciones, se puede desarrollar un servicio web compartido por todas ellas en vez de duplicar funcionalidad.
- **Gestionar iniciativas de portales**: los Servicios Web pueden ser muy útiles para gestionar y coordinar iniciativas de portales. Un portal contiene un trozo de código por cada aplicación de backend. El trozo de código se comunica con el backend y muestra la información en el portal. Los servicios web mejoran los portales de dos maneras. Por un lado entregan contenido al portal en formato XML, el cual se sencillo de mostrar. Por otro lado definen una forma sencilla en la que las piezas de código pueden acceder a las aplicaciones de backend.
- **Colaboración y compartición de información**: se hace más sencillo para los empleados colaborar y compartir información.
- **Obtención electrónica B2B (Business to Business)**: las compañías han realizado intercambios electrónicos de datos para automatizar las aplicaciones de compra durante años. Los Servicios Web pueden reducir el tiempo y coste necesarios para crear estas conexiones B2B.
- **Red de Negocio con Socios**: los Servicios Web pueden ser una gran base para la construcción de redes que permitan hacer negocios con los socios de la empresa.
- **SAS (Software-a-a-Service)**: los Servicios Web también pueden ser usados para ofrecer una interfaz de programación para servicios de negocio.

CARACTERÍSTICAS DE LOS SERVICIOS WEB

Los servicios web tiene algunas características de comportamiento especiales [27]:

- **Basados en XML**: al usar esta tecnología para la comunicación y transporte de datos se eliminan los problemas de enlazado de redes, sistemas operativos o plataformas, de esta forma se consiguen aplicaciones interoperables.
- **Débilmente acoplados**: la interfaz del servicio web pueden cambiar a lo largo del tiempo sin comprometer la habilidad del cliente para interactuar con el servicio. Esto hace que los sistemas sean más fácilmente gestionables y permite una integración más sencilla entre sistemas diferentes.
- **De grano grueso**: las tecnologías orientadas a objetos como Java ofertan los servicios a través de métodos individuales, pero estos métodos son demasiado pequeños como para ofrecer la funcionalidad necesaria. Por lo tanto se combinarán varios métodos para ofrecer un servicio de grano grueso, con su correspondiente interfaz, de forma que se pueda agrupar la cantidad necesaria de lógica de negocio.
- **Posibilidad de ser síncronos o asíncronos**: en los servicios síncronos el cliente invoca el servicio y espera su terminación, mientras que en los asíncronos una vez invocado el servicio, el cliente continúa su ejecución hasta que recibe el resultado de la invocación. Que un servicio web pueda ser asíncrono es un factor clave si se desea conseguir sistemas débilmente acoplados.
- **Soportan Invocación de procedimientos remotos (RPC – Remote Procedure Call)**: permiten a los clientes invocar procedimientos en objetos remotos usando protocolos basados en XML. Soportan el desarrollo de componentes a través de JavaBeans (EJBs) y .NET, ambas tecnologías son distribuidas y accesibles a través de mecanismos RPC.
- **Soportan el intercambio de documentos**: esta es una de las ventajas de usar XML, ya que no solo permite representar datos de forma genérica, si no también documentos, los cuales pueden ser de la complejidad deseada.

¿POR QUÉ UTILIZAR SERVICIOS WEB?

La utilización de Servicios Web tiene muchos beneficios, aquí se listan algunos de los principales [3]:

- **Exponen funcionalidad existente en la red**: un Servicio Web es un trozo de código que puede ser invocado de forma remota utilizando HTTP, por lo que se puede publicar funcionalidad ya desarrollada en la red para que pueda ser invocada por otro programa.
- **Conectan diferentes aplicaciones**: permiten a diferentes aplicaciones compartir información y servicios. Estas aplicaciones como se ha mencionado anteriormente no tiene por qué ser desarrolladas en el mismo lenguaje o estarse ejecutando sobre la misma plataforma.
- **Protocolos estandarizados**: se utilizan protocolos estandarizados en todas las capas. Esto genera una gran cantidad de ventajas de negocio como amplio rango de opciones, reducción del coste debido a la competitividad e incremento en la calidad.
- **Bajo coste de comunicación**: al utilizar SOAP sobre HTTP se puede utilizar una conexión a internet de bajo coste para realizar la comunicación. Aunque también existe la posibilidad de utilizar otros protocolos como por ejemplo FTP.

APACHE AXIS 2

Introducción

Apache Axis 2 es, junto con JAX-WS, el framework más utilizado para el desarrollo de servicios web en la plataforma Java. En las siguientes secciones se analizará este framework, los distintos procesos existentes para generar un servicio web utilizándolo y los problemas surgidos durante su uso.

Instalación

Apache Axis 2 puede ser descargado en la página oficial de la herramienta en [6]. Una vez descargado e instalado, se debe de crear una variable de entorno AXIS2_HOME que apunte al directorio donde se encuentra instalado. Además se debe tener definida la variable de entorno JAVA_HOME apuntando al directorio donde se tiene instalado el JDK.

Entre los elementos más importantes que nos podemos encontrar en el paquete están:

- **Ficheros de comandos**: que nos permitirán generar nuestro servicio web, ya sea a partir de un fichero wsdl o uno java.
- **Fichero de configuración**: para determinar el funcionamiento de Axis.
- **Librerías**
- **Aplicación Web:** Apache Axis puede ser instalado en un servidor de aplicaciones web para ser ejecutado desde un navegador.
- **Ejemplos:** distintos servicios ya implementados que se pueden instalar y ejecutar.

A continuación se analizarán más en profundidad la aplicación web y los ejemplos. Para pasar posteriormente al proceso de generación del servicio web.

Aplicación Web

Una vez instalado en un servidor de aplicaciones web, como por ejemplo Tomcat, Axis puede ser ejecutado desde un navegador proveyéndonos de las siguientes funcionalidades:

- Instalación de un servicio web a partir de un fichero con extensión "aar" (extensión del paquete que contiene el servicio web generado por Axis).
- Consulta de los servicios web actualmente disponibles, su estado, su EPR (EndPoint Reference), una breve descripción del servicio y los métodos accesibles. Así mismo se puede consultar también el fichero de especificación del servicio (wsdl) clicando en el nombre del servicio.
- También provee funcionalidad para comprobar si la instalación del propio Axis y de su aplicación web han sido correctas.

La figura 3 muestra la apariencia de la aplicación web de Axis una vez instalada, en este caso en el servidor de aplicaciones Apache Tomcat.

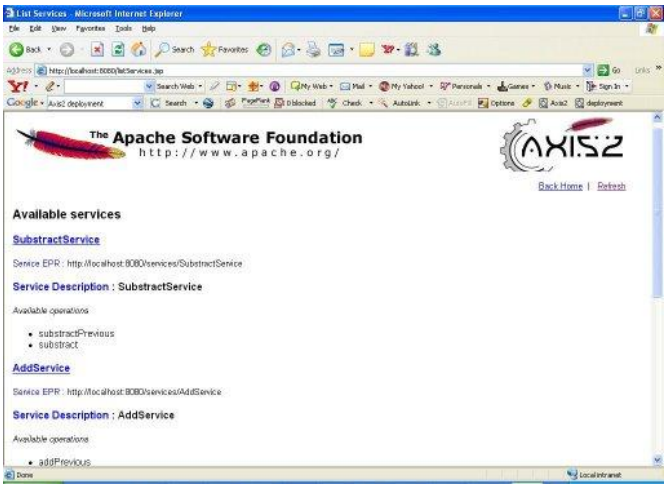

Figura 3: Apariencia de la aplicación web de Axis

Hay que mencionar que, en el caso de Tomcat es necesario modificar su configuración para permitirle la ejecución de servicios web, dichos cambios de configuración no se encuentran ni mencionados en la página oficial de Axis y es necesaria una búsqueda bastante ardua para

encontrar que cambios son estos. Además estos cambios de configuración no son de un nivel básico o medio, por lo que hay que tener bastante experiencia con Tomcat para tener conocimiento de ellos.

Ejemplos

En el directorio "samples" se pueden encontrar numerosos ejemplos de servicios web ya implementados. Lo único necesario para ejecutarlos es generar el servicio web asociado a ellos. De la misma forma algunos ejemplos también permiten la generación del código del cliente necesario para accederlos.

Como crítica personal comentar que algunos ejemplos tienen errores, ya que no es posible generar el servicio web a partir del código de ejemplo proporcionado. Así mismo, dentro de los ejemplos que incorporan la funcionalidad de generación del cliente, también existen casos en los que dicha funcionalidad no ha podido ser accedida por errores en los ejemplos. Finalmente, comentar que algunos ejemplos a pesar de la generación correcta del servicio web, una vez instalados no han proporcionado respuesta al ser invocados.

Generación del Servicio Web

Para la generación del servicio web se partió de la clase java que contiene la funcionalidad del servicio ya desarrollada. Aunque también es posible generar el servicio web a partir del fichero de descripción del servicio (wsdl) utilizando para ello el comando wsdl2java proporcionado por Axis. En este caso se explican los pasos seguidos para generar el servicio a partir de una clase java:

1. Desarrollo de la clase Java que va a implementar el servicio, en este caso particular se utilizó el IDE Eclipse
2. Escribir el fichero build.xml. Este fichero es el encargado de ejecutar las distintas tareas necesarias para llevar a cabo tanto la generación de la descripción del servicio, el empaquetado del mismo, así como la generación del código del cliente. Este fichero será interpretado por Apache Ant (que puede ser descargado en [26]) para ejecutar las tareas anteriormente mencionadas. En el Anexo 1 se puede encontrar el fichero build.xml utilizado.
3. Compilación del código Java desarrollado. Para ello se utilizará Ant y la tarea definida en build.xml. Esta acción provocará la generación del directorio "build" dentro de la carpeta del proyecto. La figura 4 muestra la estructura de este directorio[1]. Dentro nos encontramos la carpeta "classes", la cual almacenará las clases Java una vez compiladas. También podemos observar que es en este directorio donde se almacenara el fichero wsdl que se genera en el paso 4 (HouseFacade.wsdl), así como la aplicación una vez empaquetada que se genera en el paso 5 (HouseFacade.aar).

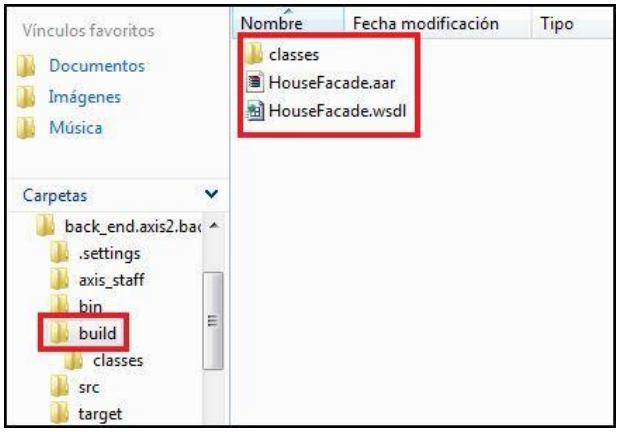

Figura 4: Estructura del directorio build

4. Generación del fichero de descripción del servicio (wsdl), utilizando para ello Ant y la tarea definida en el fichero build.xml.
5. Empaquetado del servicio web, en este caso la extensión del paquete es .aar[2].
6. Instalación del servicio web (fichero con extensión .aar) en el directorio de servicios de Axis[3]. Dicho directorio se muestra en la figura 5.

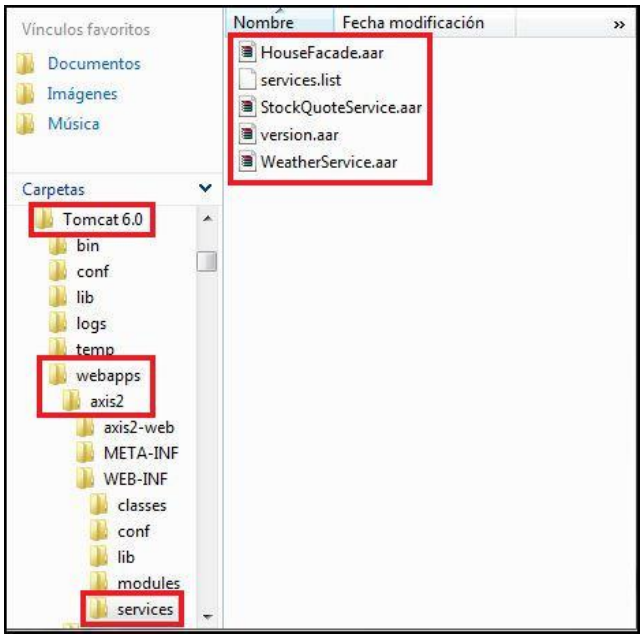

Figura 5: Directorio de la aplicación web de Axis

[1] Esta es la estructura definida por defecto en los ejemplos de Axis. Si bien esta estructura, así como el nombre de las carpetas puede ser modificada al gusto del desarrollador cambiando el fichero build.xml

[2] Esta es la extensión utilizada por defecto por Axis para empaquetar los servicios. No se ha probado a utilizar otras extensiones con Axis, no obstante se sabe que se pueden utilizar otros formatos de empaquetamiento para servicios web como el .war.

[3] También existe la posibilidad de utilizar la interfaz de la aplicación web de Axis para instalar el fichero empaquetado.

Una vez instalado nuestro servicio web, este será accesible en (1)[4]

http://localhost:8080/xxx/yyy (1)

Siendo "xxx" el nombre del paquete y "yyy" el nombre del servicio. Así mismo la definición del servicio en formato wsdl podrá ser consultada en (2)

http://localhost:8080/xxx/yyy?wsdl (2)

Como se puede comprobar el proceso de generación de un servicio web es rápido y sencillo. Además es independiente de la cantidad y complejidad de los métodos que el servicio proporcione, ya que este factor solo afectará al tiempo necesario para el desarrollo de la clase Java que implemente el servicio, siendo necesario siempre el mismo tiempo para la compilación del código Java, la generación del fichero wsdl, el empaquetado de la aplicación y su instalación.[5]

Actualmente no se encuentra ningún manual en la página oficial de Axis sobre la estructura y los elementos del fichero build.xml. Por tanto es necesario invertir una serie de horas al comienzo de la utilización de Axis para su comprensión. El conocimiento de dicho fichero se basa, por tanto, en la observación de los ejemplos y las deducciones que cada uno pueda hacer sobre el significado y estructura de cada uno de sus elementos y sus parámetros. Una vez conseguido el conocimiento suficiente es cuando uno está listo para crear el fichero build.xml de su servicio web.

Generación del cliente

Una vez hemos generado e instalado el servicio web, es hora de crear el código cliente que nos permita invocar dicho servicio. Para ello una vez más utilizaremos el fichero build.xml conjuntamente con ant para que Axis nos genere las clases básicas del cliente. Una vez disponemos de estas clases debemos generar nuestra propia clase cliente principal, en la que especificaremos el punto de acceso y la cual se encargará de invocar los diferentes métodos del servicio web. En la página web de Axis [6] se pueden encontrar ejemplos sencillos del código a desarrollar en el cliente.

Integración

Actualmente al igual que muchos proyectos desarrollados por Apache, Axis tiene integración con el IDE Eclipse. Otro proyecto desarrollado por Apache es el servidor de aplicaciones Tomcat, el cual también posee integración con Eclipse. De esta forma si se configura Eclipse de la forma adecuada se podrá utilizar el "wizard" que se observa en la Figura 6 para la generación de Servicios Web

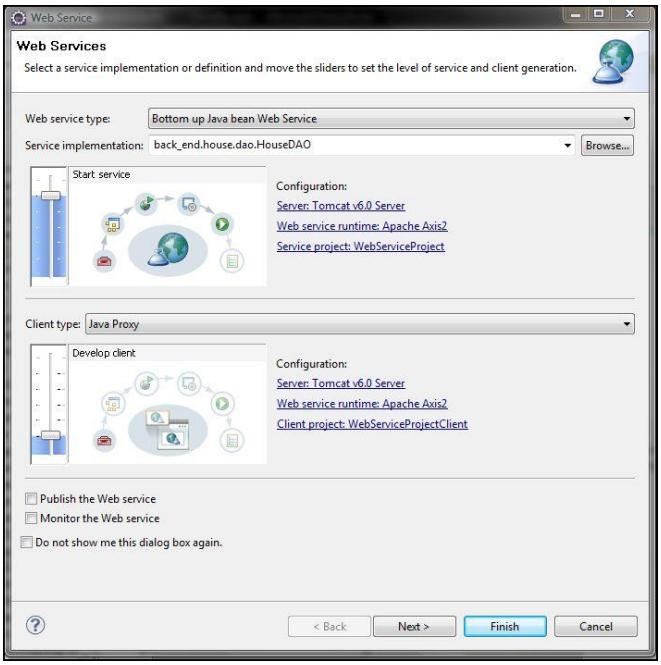

Figura 6: Wizard para generación de Servicios Web de Eclipse

Este "wizard" nos permite generar a partir de la clase Java del Servicio Web, el fichero wsdl de especificación del servicio web, así como las clases cliente. También existe la posibilidad de generar la implementación del servicio a partir de su especificación. En este caso a partir de un fichero wsdl se podrá publicar el servicio, generar la clases cliente y el esqueleto de la clase Java que implemente el serivicio.

El "wizard" también nos permite especificar hasta qué punto del ciclo de vida de desarrollo de un servicio web queremos que sea realizado el proceso de forma automática. Así pues tanto para la generación del servicio como de las clases cliente podemos seleccionar desde "Develop" hasta "Test" donde el servicio ya ha sido desarrollado, instalado en un servidor de aplicaciones y está listo para ser probado. Las fases que pueden ser seleccionadas son las siguientes:

1. Develop
2. Assemble
3. Deploy
4. Install
5. Start
6. Test

[4] Para el ejemplo mostrado en las Figuras 4 y 5 la URL sería http://localhost:8080/HouseFacade/HouseFacade.

[5] De hecho en el fichero build.xml se puede crear una tarea que agrupe a todas estas, siendo necesario la utilización de una única orden en la línea de comandos.

Como experiencia personal comentar que no he sido capaz de llegar a utilizar dicho "wizard" debido a que la integración de Tomcat con Eclipse me generaba problemas, puede ser porque estuviera mal configurada. Eclipse me detectaba Tomcat como servidor de aplicaciones instalado y funcionando, pero a la hora de instalar el servicio utilizando este "wizard" se generaba un error. Ya que el proceso de instalación es completamente independiente de Axis, deduzco que se debe tratar de un problema de integración con Tomcat. Descarto también la posibilidad de que hubiera un problema en el servidor de aplicaciones porque este funcionaba perfectamente para ejecutar otras aplicaciones ya instaladas.

Axis dispone a su vez de integración con otros productos desarrollados por Apache como por ejemplo el gestor de proyectos Maven [28]. Este gestor funciona en base a un fichero de descripción del proyecto llamado pom.xml, en el cual se pueden especificar dependencias del proyecto de forma que el propio programa se encargue de descargar las librerías necesarias y llevar a cabo algunas fases del ciclo de vida de forma automática.

En mi caso particular, encontré publicadas en la web la especificación de las dependencias de Axis que se deben incluir en el fichero pom.xml. Ahora bien, a la hora de ejecutar Maven para que descargara las librerías de Axis necesarias para la generación del servicio web, este informaba de un error indicando que las librerías de Axis no se encontraban en el repositorio, por lo tanto tampoco pude generar el servicio web de forma automática con esta herramienta.

Valoración personal

Apache Axis 2 es una de los dos framework más utilizados en el mercado para el desarrollo de servicios web sobre la plataforma Java. Se entiende por qué, ya que es una herramienta potente, que permite generar de forma automática tanto el código del cliente como el fichero de especificación del servicio (wsdl).

Otra de sus grandes ventajas es que al ser desarrollado por Apache se puede integrar con otras de sus herramientas como Apache Tomcat o Apache Maven; además teniendo en cuenta la predilección que desde el proyecto Apache se tiene por el IDE Eclipse actualmente existen plugins para dicho IDE que permiten integrar la funcionalidad de Axis con Tomcat y con el propio IDE, poniendo la generación de servicios a unos simples clicks de distancia. El hecho de que exista este nivel de integración lleva a Axis un paso más allá que el otro framework estudiado JAX-WS haciendo que el proceso de generación sea más sencillo, más rápido y que no sea necesario utilizar la línea de comandos, lo que puede ser una gran ventaja para los desarrolladores primerizos. Actualmente esta es la mejor y más eficiente forma que se ha encontrado para desarrollo de servicios web.

La gran desventaja de Axis reside en su falta de estabilidad. Con esto se hace referencia a que se han tenido gran cantidad de problemas para la utilización de esta herramienta. En primer lugar si no se utiliza ninguna de las opciones de integración es necesario pasar varias horas estudiando los ficheros build.xml de los ejemplos para conseguir entender su estructura y el significado de sus campos, para así poder generar uno propio. Una vez superado este problema y con el servicio web ya generado, es necesario cambiar la configuración de Tomcat para que este sea capaz de soportar servicios web, la búsqueda de dicha configuración fue muy larga ya que en la web se encuentran muy pocas personas que se hayan encontrado con este problema y hayan sido capaces de resolverlo. Aun así solo se ha sido capaz de instalar el servicio web, su ejecución nunca se ha podido llevar a cabo. Para la generación del cliente el principal problema reside en que hay muy pocos ejemplos. De esta forma es difícil cotejar como es la especificación de dicha generación en el fichero build.xml.

Respecto a las posibilidades de integración, estas parecen ser el uso futuro de la herramienta, dejando atrás la especificación de ficheros y la utilización de la línea de comandos. Pero actualmente no se ha sido capaz de llevar a cabo la generación de servicios utilizando dichas funcionalidades de integración.

Referente a los ejemplos, hay que mencionar que la herramienta incorpora una gran cantidad de ellos en el paquete de descarga, si bien no todos parecen ser correctos. Se intentaron generar, instalar y probar algunos de ellos, obteniendo diferentes resultados para cada caso. La generación siempre fue posible, pero la instalación no lo fue en todos los casos; además otros fallaban en la generación del código cliente.

Como valoración final comentar que Apache Axis 2 es una herramienta con una gran funcionalidad y muchas posibilidades de cara al futuro gracias a sus capacidades de integración, pero actualmente la herramienta no es lo suficientemente estable como para su uso.

JAX-WS

Introducción

Actualmente JAX-WS es uno de los framework más utilizados para el desarrollo de servicios web en la plataforma Java. A continuación se analizará el proceso de generación y consumo de servicios utilizándolo.

Instalación

El paquete que contiene JAX-WS puede ser descargado de la página oficial, una vez instalado es necesario modificar la variable de entorno CLASSPATH para que incluya los comandos de JAX.

En caso de que se vaya a utilizar GlassFish como servidor de aplicaciones, la descarga de este es suficiente, al ya traer incorporados los comandos de JAX-WS necesarios para la generación de servicios.

Generación del Servicio Web

El primer paso para la generación del Servicio Web es desarrollar la clase Java encargada de ejecutar el servicio, para ello se deben tener en cuenta las siguientes consideraciones:

- **Constructor por defecto**: toda clase que vaya a contener métodos invocables por servicio web o que vaya a ser utilizada debe incorporar el constructor por defecto sin parámetros.
- **Anotaciones**: la clase que contenga el servicio web debe ser anotada con "@WebService" y los métodos que vayan a ser invocables deben ser anotados con "@WebMethod".
- **Restricciones de métodos**: los métodos que vayan a ser invocables no pueden ser estáticos ni privados. La clase que implementa el servicio web puede tener métodos con estas características, pero estos no deberán ser invocables.
- **Restricciones de clase**: la clase que implemente el servicio web no puede ser abstracta ni final.

En el Anexo 2 se puede encontrar el código de una clase que implementa un servicio web. Se ha utilizado una clase muy sencilla como ejemplo del proceso de generación de servicio ya que dicho proceso es independiente de la complejidad o tamaño de la clase Java. El proceso de generación será y durará lo mismo tenga la clase de implementación un solo método o 20 métodos extremadamente complejos. Se ha escogido esta orientación ya que la finalidad es mostrar el proceso de generación, no el desarrollo de métodos Java. Además la sencillez del ejemplo hace que este sea más fácilmente comprensible.

Una vez se ha desarrollado la clase Java se utilizará el comando "wsgen" para la generación del fichero de especificación del servicio (wsdl). Dicho comando recibe como parámetro la clase Java que implementará el servicio.

Ahora que ya se tiene generado el fichero wsdl se debe cambiar la configuración de la aplicación para indicar aspectos como que se debe ejecutar como servicio web, así como el nombre por el que será accesible. Esto se consigue modificando el fichero web.xml y creando el fichero sun-jaxws.xml. El contenido de dichos ficheros para el ejemplo que se está tratando se puede encontrar en los Anexo 3 y Anexo 4 respectivamente.

Una vez realizado la configuración ya se tiene todo listo para poder instalar el servicio web. El siguiente paso será empaquetar el proyecto, para ello en este caso se ha utilizado el comando "package" de Apache Maven que compilará las clases y empaquetará el proyecto en un fichero con extensión .war[6].

Por último se debe instalar el servicio web. Para la realización del ejemplo se ha utilizado GlassFish Server[7], por tanto una vez arrancado el servidor con el comando

asadmin start-domain

se utilizará el siguiente comando para instalar el servicio web

asadmin deploy XXX

donde XXX es el servicio web una vez empaquetado, en este caso wsSample.war.

Una vez terminados estos pasos el servicio estará disponible en la dirección (3)

http://localhost:8080/wsSample/wsSample (3)

Así mismo la definición del servicio en formato wsdl podrá ser consultada en (4)

http://localhost:8080/wsSample/wsSample?wsdl (4)

Generación del Cliente

Una vez el servicio web ha sido instalado en un servidor de aplicaciones y es accesible a través de la web, el siguiente paso es generar el código cliente que se encargue de accederlo. Para ello se utilizará el comando "wsimport", dicho comando recibe como parámetro el fichero wsdl de especificación del servicio, que puede ser una URL a dicho fichero o una ruta local[8]. Una vez ejecutado este comando generará las clases Java necesarias para invocar el servicio que en dicho fichero wsdl se encuentran especificadas.

[6] Existen otros procedimientos para el empaquetado de proyectos, pudiéndose utilizar el IDE Eclipse, así como la línea de comandos. De la misma forma también se pueden utilizar otros formatos de empaquetado diferentes al war.

[7] En caso de utilizar un servidor de aplicaciones diferente el proceso debería ser el mismo que el de instalación de cualquier otra aplicación, pero puede que su servidor no soporte Servicios Web o que necesite de configuración adicional. En caso de duda consulte la documentación de su servidor de aplicaciones.

[8] El comando "wsimport" admite la especificación de otros parámetros adicionales.

Una vez disponemos de estas clases el siguiente paso será codificar nuestro cliente, en el Anexo 5 se puede encontrar el código ejemplo de un cliente que accederá al servicio que hemos generado en el apartado anterior.

Con el código cliente ya escrito, el último paso será ejecutar dicho cliente para comprobar que este, así como el servicio web, funcionan[9].

Documentación y Ejemplos

Actualmente se pueden encontrar en la web gran cantidad de tutoriales sobre cómo crear servicios web utilizando JAX-WS, pero lo que hay que destacar es la falta de homogeneidad del proceso, siendo que cada tutorial especifica una forma parecida, pero a la vez distinta a la de todos los demás.

También hay que mencionar que algunos de estos tutoriales incluyen errores en su código de ejemplo, como en el principalmente utilizado para el desarrollo del código cliente [9], donde no se instancia el servicio.

Se debe decir también que aunque algunos de estos tutoriales son genéricos, de forma que no se especifica un servidor de aplicaciones o herramientas a utilizar, otros muchos son dependientes de una o varias herramientas, de forma que el proceso es solo válido para estas. Especialmente remarcable es el caso de utilizar Netbeans como IDE y GlassFish como servidor de aplicaciones, además del ya mencionado en el apartado de Axis, Apache Ant.

Valoración Personal

JAX-WS es uno de los frameworks más utilizados para el desarrollo de Servicios Web con la plataforma Java, además de ser el sobre el que más documentación se encuentra en la web.

Su gran valor reside en la simplicidad de su uso. En primer lugar los comandos necesarios para ejecutarlo vienen ya incorporados en GlassFish por lo que se hace muy sencillo tener acceso a toda la plataforma necesaria para poder desarrollar con esta tecnología. Y en segundo lugar el proceso de generación del servicio es extremadamente sencillo, ya que solo es necesario ejecutar dos comandos y especificar dos ficheros de configuración, los cuales son sencillos de entender y en los que solo se deben modificar un par de campos para especificar el nombre que se va a dar al servicio e indicar cuál será la clase que lo implemente.

[9] En esta sección se han explicado los pasos para llevar a cabo la generación del cliente de la forma más sencilla posible, pero existen otras posibilidades en las cuales se puede observar el código generado por el comando, así como editarlo, para que por ejemplo nos permita utilizar algún programa como Membrane [13] para observar el paso de mensajes SOAP.

No obstante si se desea hace un uso más avanzado de estos comandos se debe tener cuidado, ya que en algunos casos como en el que se indica que se mantengan las clases de código generadas para la creación del cliente, si no se maneja con cuidado, la ejecución del comando podría borrar algunos directorios de código sin avisarnos previamente de tal peligro.

También hay que destacar que cuando se indica que se mantengan las clases Java generadas por el comando "wsimport", Eclipse encuentra errores en el código de estas clases provocando errores de compilación, sin embargo cuando el comando genera directamente los archivos .class esto no ocurre.

La principal desventaja de este framework es el hecho de que casi todos los tutoriales que se encuentran en la red son dependientes de alguna herramienta, de forma que limitan la libertad de decisión a la hora de escoger con que programas trabajar.

COMPARATIVA DE LAS DOS HERRAMIENTAS

Actualmente estos dos frameworks son los más utilizados del mercado para desarrollo de Servicios Web con Java y se entienden los motivos de tal situación. Ambos proveen formas muy rápidas y sencillas de generar un servicio web, si bien cada uno de forma diferente.

Ambos, en su forma más básica se basan en la utilización de la línea de comandos para todo el proceso de generación del servicio. Pero mientras Axis ha desarrollado opciones de integración con otras herramientas de forma que este proceso se hace aún más sencillo, JAX-WS no ha implementado tal funcionalidad.

Cada herramienta parece estar asociada a un gestor de aplicaciones e IDE diferente y por tanto siguen caminos distintos. Por un lado Axis es desarrollado por el proyecto Apache, el cual dispone de Tomcat como servidor de aplicaciones y por tanto con el que se integra a través del IDE Eclipse, el cual también es el principalmente utilizado por las herramientas de Apache. Por otro lado JAX-WS es incorporado dentro del servidor de aplicaciones GlassFish y en las páginas de ambos productos los tutoriales hacen referencia a la utilización de Netbeans como IDE. Por tanto parece que se establecen dos arquitecturas de aplicaciones diferentes según que herramienta se desee usar, teniendo así por un lado Netbeans – JAX-WS – GlassFish y por el otro Eclipse – Axis 2 – Tomcat. No obstante mencionar que para el desarrollo el ejemplo presentado en la sección de JAX-WS se utilizó Eclipse con JAX-WS y GlassFish.

La gran cantidad de documentación encontrada en la web hace quizás inclinarse por JAX-WS al tener más

tutoriales de su uso e instalación. Además en algunos lugares como [29] se recomienda la utilización de JAX-WS sobre Axis.

De cara al futuro se considera que la mejor elección es la de Apache Axis 2 y he aquí los principales motivos para tal elección:

- Actualmente el uso de Eclipse está más extendido que el de Netbeans, además existen una gran cantidad de plugins para integrar distintas herramientas y funcionalidades dentro del IDE, mientras que para Netbeans, si bien existen plugins, su número no es tan cuantioso.
- La gran experiencia en desarrollo de herramientas de la que provee el proyecto Apache. Con esto no se trata de menospreciar a los desarrolladores de JAX-WS pero el proyecto Apache lleva muchos años establecido como desarrollador de software.
- Finalmente y como punto más importante, Axis provee de unas opciones de integración muy interesantes, como lo son con el gestor de proyectos Apache Maven y con el duo Apache Tomcat-Eclipse. Estas posibilidades de integración representan un avance importante al eliminar la interacción del programador con la línea de comandos, lo cual puede resultar muy ventajoso para los usuarios más inexpertos. Además del hecho de reducir cuantiosamente el tiempo necesario para la generación del servicio y del código del cliente. Si bien este tiempo ya era poco y el proceso sencillo, con la integración con Tomcat y Eclipse se reduce aún más.

DESARROLLO DE SERVICIOS WEB ASÍNCRONOS

Con los servicios web asíncronos el cliente invoca un servicio pero no puede esperar a que este le responda, por ejemplo porque esta respuesta puede tardar mucho tiempo en llegar. En estos casos el cliente continúa su ejecución hasta que recibe el resultado de la invocación del servicio, que a continuación pasaría a tratar. En [5] se especifican una serie de posibilidades para solventar esta situación.

La primera alternativa consiste en una aproximación basada en documentos. En este caso el cliente especifica en un documento las acciones que desea que lleve a cabo el servidor y este documento es la entrada al servicio web.

La segunda consiste en desarrollar una capa intermedia entre el consumidor y el proveedor del servicio. Cuando el consumidor desea invocar el servicio este se pone en contacto con la capa intermedia, la cual le retorna un ID. Posteriormente la capa intermedia se encarga de gestionar la invocación del servicio. Una vez llegados a este punto existen dos alternativas:

- El cliente consulta periódicamente el estado de su petición utilizando para ello el ID proporcionado por la capa intermedia.
- En caso de que el cliente sea a su vez un servicio web, el proveedor de servicio puede aprovecharse de esta situación para comunicarle a través de su interfaz el resultado de la invocación.

CONCLUSIONES

Los Servicios Web son una de las tecnologías más establecidas y utilizadas actualmente. Gran cantidad de organizaciones utilizan actualmente esta tecnología, por ello se hace indispensable adquirir conocimientos sobre su arquitectura, usos, desarrollos…

Su arquitectura está estructurada en diversas capas, en las cuales se utilizan protocolos estandarizados. Estas capas nos proporcionan una amplia funcionalidad principalmente en los niveles superiores (composición de servicios, especificaciones adicionales).

Los servicios web ponen a nuestra disposición una amplia gama de posibilidades tales como publicar funcionalidad ya desarrolladas, independizarnos de la plataforma y/o del lenguaje de implementación de forma que aplicaciones heterogéneas puedan comunicarse y hasta componerse para formar unas nuevas más complejas. Especialmente interesantes son las especificaciones adicionales las cuales nos proveen de una gran cantidad de posibilidades para ofrecer distintas calidades de servicio.

Se han estudiado los procesos de generación de Servicios Web con los dos framework de desarrollo Java más utilizados del mercado, pudiendo comprobar lo sencillo y rápido que es el proceso de desarrollo, poniéndolo así al alcance de casi cualquier programador, no siendo necesario un gran nivel ni experiencia.

En definitiva, los Servicios Web son una tecnología muy conocida y utilizada, con muchas posibilidades de futuro, que cualquier persona del ámbito de las tecnologías de la información debería conocer y saber utilizar.

LÍNEAS FUTURAS

En este apartado se comentan algunas líneas futuras de investigación que se han excluido del documento al considerar que se desvían del tema central. A continuación se mencionan las más importantes:

- **Análisis en mayor detalle de SOAP, WSDL y UDDI**: son los tres protocolos que forman la base de la especificación de Servicios Web. Por su importancia un estudio en mayor profundidad podría resultar de interés.

- **Análisis en mayor detalle de las especificaciones adicionales**: son uno de los puntos fuertes de los Servicios Web al permitir la especificación de diferentes calidades de servicio. Estas especificaciones incorporan nuevas y muy interesantes funcionalidades a los Servicios Web.
- **Análisis de herramientas disponibles para otros lenguajes de programación**: ya que Java no es el único lenguaje de programación en el mercado, también es interesante hacer un análisis de cuáles son las herramientas disponibles para hacer desarrollos con otros lenguajes de programación y cuáles son sus características.

REFERENCIAS

[1] Historia de los Servicios Web
http://www.w3.org/2002/ws/history.html

[2] "Web Services: A Manager's Guide". Anne Thomas Manes.2004. Pearson Education
http://www.computerworld.com/s/article/94886/Book_Excerpt_When_to_Use_Web_Services

[3] ¿Por qué utilizar Servicios Web?
http://www.tutorialspoint.com/webservices/why_web_services.htm

[4] Tutorial de Servicios Web en la página del W3C
http://www.w3schools.com/webservices/default.asp

[5] Servicios Web asíncronos
http://java.sun.com/blueprints/webservices/using/webservbp3.html

[6] Página web de Apache Axis 2
http://axis.apache.org/axis2/java/core/index.html

[7] Tutorial de JAX-WS
http://tundidor.com/blog/?p=78

[8] Página web de JAX-WS
http://jax-ws.java.net/

[9] Tutorial de Java EE 6: Servicios Web con JAX-WS
http://download.oracle.com/javaee/6/tutorial/doc/bnayn.html

[10] Tutorial de JAX-WS
http://www.myeclipseide.com/documentation/quickstarts/webservices_jaxws/

[11] Tutorial de JAX-WS
http://www.java-tips.org/java-ee-tips/java-api-for-xml-web-services/developing-web-services-using-j.html

[12] Tutorial de JAX-WS
http://www.javadb.com/create-a-web-service-client-with-jax-ws

[13] Página web de Membrana SOAP Monitor
http://www.membrane-soa.org/soap-monitor/

[14] IEEEXplore. Página de papers de IEEE
http://ieeexplore.ieee.org/Xplore/guesthome.jsp

[15] Historia de los Servicios Web
http://www.xml.com/pub/a/ws/2001/04/04/soap.html

[16] Servicios Web en la Wikipedia
http://en.wikipedia.org/wiki/Web_service

[17] Comentarios sobre Servicios Web
http://www.w3schools.com/webservices/ws_why.asp

[18] Página oficial de Servicios Web
http://www.webservices.org/

[19] "Web Services Technologies: State of the Art. Definitions, Standards, Case Study". Abdalhem Albreshne, Patrik Fuhrer, Jacques Pasquier. 2009

[20] IBM. WebSphere Business Integration Adapters.
http://publib.boulder.ibm.com/infocenter/wbihelp/v6rxmx/index.jsp?topic=/com.ibm.wbia_adapters.doc/doc/webservices/webservices17.htm

[21] "Web Services Computing: Beyond Component-Based Architectures". Dionisis X. Adamopoulos, Constantine A. Papandreou.

[22] "Web Services Architectures". Judith Myerson. 2009

[23] "Web Services: Principles and Technology" Michael Papazoglou. 2008.

[24] Lista de especificaciones adicionales de los servicios web
http://en.wikipedia.org/wiki/List_of_web_service_specifications

[25] "Web Services: Introduction and State of The Art". Óscar Corcho García, José Carlos del Arco Prieto, Jesús Arias Fisteus

[26] Página Web de Apache Ant
http://ant.apache.org/

[27] Características de los Servicios Web
http://www.tutorialspoint.com/webservices/web_services_characteristics.htm

[28] Página Web de Apache Maven
http://maven.apache.org/

[29] Comentarios de comparativa entre Axis y JAX-WS
http://tundidor.com/blog/?p=22

ANEXOS

Anexo 1: Fichero build.xml

```
<project name="quickstart" basedir="." default="generate.service">

   <property environment="env"/>
   <property name="AXIS2_HOME" value="C:\Program Files\apache-axis2-1.5.2"/>

   <property name="build.dir" value="build"/>
         <property name="tomcat.repository" value="C:\Program Files\Apache Software Foundation\Tomcat
6.0\webapps\axis2\WEB-INF\services" />

   <path id="axis2.classpath">
      <fileset dir="${AXIS2_HOME}/lib">
         <include name="*.jar"/>
      </fileset>
                  <fileset dir="C:\Users\abraham\.m2\repository">
                        <include name="**/*.jar"/>
                  </fileset>
   </path>

   <target name="compile.service">
      <mkdir dir="${build.dir}"/>
      <mkdir dir="${build.dir}/classes"/>

      <!--First let's compile the classes-->
      <javac
          destdir="${build.dir}/classes"
          srcdir="${basedir}/src/main/java"
          classpathref="axis2.classpath"
                              includes="utility/**, back_end/**">
      </javac>
   </target>

   <target name="generate.wsdl" depends="compile.service">
      <taskdef name="java2wsdl"
            classname="org.apache.ws.java2wsdl.Java2WSDLTask"
            classpathref="axis2.classpath"/>
      <java2wsdl className="back_end.house.facade.HouseFacade"
             outputLocation="${build.dir}"
             targetNamespace="http://facade.house.back_end/"
             schemaTargetNamespace="http://facade.house.back_end/xsd">
         <classpath>
            <pathelement path="${axis2.classpath}"/>
            <pathelement location="${build.dir}/classes"/>
         </classpath>
      </java2wsdl>
   </target>
```

```
   <target name="generate.service" depends="generate.wsdl">
      <!--aar them up -->
      <copy toDir="${build.dir}/classes" failonerror="false">
         <fileset dir="${basedir}/src/main/resources">
            <include name="META-INF/*.xml"/>
         </fileset>
      </copy>
                  <copy file="${build.dir}/HouseFacade.wsdl" tofile="${build.dir}/classes/META-INF/HouseFacade.wsdl"
overwrite="true" />
      <jar destfile="${build.dir}/HouseFacade.aar">
         <fileset excludes="**/Test.class" dir="${build.dir}/classes"/>
      </jar>
   </target>

         <target name="install" depends="generate.service">
                  <copy file="${build.dir}/HouseFacade.aar" tofile="${tomcat.repository}/HouseFacade.aar"
overwrite="true" />
                  <copy file="${build.dir}/HouseFacade.aar"
tofile="${AXIS2_HOME}/repository/services/HouseFacade.aar" overwrite="true" />
         </target>

   <target name="clean">
      <delete dir="${build.dir}"/>
   </target>
</project>
```

Anexo 2: Ejemplo de clase que implementa Servicios Web

```
package wsSample;

import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
public class WSSample {

         public WSSample(){};

         @WebMethod
         public String hello(String name) {
                  return "Hello" + name + ", welcome to web services";
         }
}
```

Anexo 3: Especificación del fichero web.xml

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
```

```
	<display-name>Archetype Created Web Application</display-name>

	<listener>
		<listener-class>
			com.sun.xml.ws.transport.http.servlet.WSServletContextListener
		</listener-class>
	</listener>

	<servlet>
		<servlet-name>wsSample</servlet-name>
		<display-name>wsSample</display-name>
		<description>An example of Web Services</description>
		<servlet-class>
			com.sun.xml.ws.transport.http.servlet.WSServlet
		</servlet-class>
		<load-on-startup>1</load-on-startup>
	</servlet>

	<servlet-mapping>
		<servlet-name>wsSample</servlet-name>
		<url-pattern>/wsSample</url-pattern>
	</servlet-mapping>

</web-app>
```

Anexo 4: Especificación del fichero sun-jaxws.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime'
	version='2.0'>
	<endpoint name='wsSample' implementation='wsSample.WSSample'
		url-pattern='/wsSample' />
</endpoints>
```

Anexo 5: Ejemplo de clase cliente que accede al servicio web

```
package client;

import generatedClient.wssample.WSSample;
import generatedClient.wssample.WSSampleService;

import javax.xml.ws.WebServiceRef;



public class WSClient {

	@WebServiceRef(wsdlLocation =
		"http://localhost:8080/wsSample/wsSample?wsdl")
	private static WSSampleService service = new WSSampleService();
	private static WSSample port =
		(WSSample) service.getWSSamplePort();

	public static void main (String [] args) {
```

```
                System.out.println(port.hello("Andrea"));
        }
}
```